

## Position Paper:

### New Mathematics for Exascale Computational Science?

Ulrich Ruede, University of Erlangen, [ruede@cs.fau.de](mailto:ruede@cs.fau.de)

May 7, 2013

Let's start with the good news first: Mathematics has been and continues to be the **most important contributor** to any large scale computational science. This is simply so, since computational complexity becomes ever more important with faster computers. Once the systems are large enough, the algorithms with a better asymptotic complexity will always make the winner. Math journals, such as SINUM or SISC (for which I had served as EIC) are full of great papers advancing computational science.

However, having said this, we find that these novel algorithms often **underperform by many orders of magnitude**. Different from what may be a belief in parts of the math community, it is not that we just need a few extra weeks for converting matlab to Fortran and MPI. Designing efficient **HPC** software requires much more **creative research** and the **deficiencies** are much more **fundamental**.

- Nothing is more practical than a **good theory**, but in the math community there is a deeply rooted **misconception about the role of rigorous theory**. For example, a rigorous asymptotic bound of the form  $|e| \leq C h^p$  has only heuristic implications when assessing the quality of a discretization for all finite values of  $h$  (that is in all practical computing). Such theorems are a poor basis for comparing one discretization to another one of the same or even a different order, as long as the constants remain unspecified. We need more **quantitative theory**. Where this is not available, systematic numerical experiments are as important or even more important than rigorous theory. Some fields of contemporary applied math have an underdeveloped tradition in **systematic algorithmic benchmarking**. This starts with a lack of generally accepted standard test examples. Therefore, the numerical cost of an algorithm (i.e. the number of flops induced by using a specific discretization or by a specific solver) is frequently left unquantified. Consequently rather inefficient algorithms remain in use even when better alternatives exist.

- On modern computer systems, the traditional **cost metric of numerical mathematics** (i.e. the Flops for solving a problem) **fails increasingly to correlate** with the truly relevant cost factors, such as **time to solution** or **energy consumption**. It will be necessary to quantify much more complex algorithmic characteristics, such as memory footprint and memory access structure (i.e. cache re-use, uniformity of access, utilization of block-transfers, etc) processor utilization, communication and synchronization requirements. These effects must be built into **better complexity models** – models that are simple enough that they can be used, but that capture the true nature of computational cost much better than just counting the Flops.

- For exascale computational science we need to develop a more systematic integrated **algorithm engineering methodology**. Starting from the mathematical model, we should **predict a-priori** what performance is achievable, and then we must evaluate our realization with respect to such a prediction, **accounting for all the discrepancies**. This must be done **on all levels** of the “simulation pipeline”, starting with the mathematical model, the discretization, the solver, its sequential implementation, and eventually its parallelization. It is essential that this is not just seen as tweaking a given algorithm to run fast on particular architecture, but as a true **co-design**. This in particular includes the design and development of the algorithms and data structures. For example, when it is known that a 2D multigrid Poisson solver reaches  $h^2$ -discretization accuracy in less than 30 operations per unknown, then we must justify the use of a more complicated discretization and more expensive solver for the same problem class. There may be good

reasons, but such **algorithmic choices** must be based on clear arguments **accounting for the accuracy achieved relative to the cost**.

- On the **implementation** side, often even the sequential version of an algorithm reaches only a fraction of the peak performance of a core. We should justify why this is the case. For example, we may find that the memory or communication bandwidth is the relevant bottleneck. Generally, theory must guide us what **bounds for the hardware performance** we must expect, and the design process must be based on a systematic accounting for the limiting resources. For this it is essential to have **realistic a-priori cost predictions** everywhere in the development process. And quite generally, we should be more honest when assessing parallel performance. “David Bayley’s “**Twelve Ways for Fool the Masses ...**”<sup>1</sup> are still too much in use.

This list of deficiencies provides already enough problems for a multi-decade math research program, but beyond there are also **great opportunities** for novel mathematical research directions. I’ll mention a few:

- With  $10^9$  parallel threads (in a future exscale system) we must avoid all unnecessary communication and synchronization. While research has already started in fields such as dense linear algebra, this is wide open elsewhere, e.g. for iterative solvers. New **asynchronous, communication avoiding** algorithms must be designed. Lower bounds must be found on how much communication/synchronization is necessary to solve a particular problem. Chaotic relaxation strategies or **stochastic and nondeterministic algorithms** may become a key innovation needed for exascale, and they may at the same time provide for more robustness and built-in **fault tolerance** overall.

- Exascale will provide the computational power to go from qualitative simulation to predictive simulation, and from predictive simulation to optimization, **parameter identification, inverse problems**; it will enable **stochastic** simulations and to better **quantify uncertainties**.

- Exascale enables us to bridge physically **from the meso-scale to human scale**. With mesa-scale, I mean here physical scales such as the cell of a biological system, a particle in a pile of sand, or a pore in an aquifer. A living human has around  $10^{11}$  neurons and  $10^{13}$  red blood cells, a pile of sand may have  $10^{10}$  grains. The mesoscale is halfway between atomic scale and human scale. Mesoscale requires to **deal with large numbers of objects**, but such ensembles may become tractable on exascale systems, since with  $10^{18}$  Flop/s we can still perform  $O(10^5)$  Flop/s for each human blood cell per second. Thus exascale may offer new possibilities for simulation science that have been out of reach without. However, to exploit this, new methods must be devised to model and simulate large mesoscopic ensembles for long enough times. New algorithms must be invented for this, **new modeling paradigms** devised. New techniques for **validation and verification** are needed: we are not interested to predict each individual blood cell in a human being accurately, but the ensemble behavior must be physically meaningful and must provide e.g. physiological insight beyond classical techniques. There are close relations here to **multiscale-modeling and multiphysics**, but these fields gain new momentum with the advent of exascale. A myriad of interesting and open research topics can be derived from this.

**Summarizing:** I believe that the advent of exascale forces mathematics to address the **performance abyss** that widens increasingly between existing math theory and the practical use of HPC systems. Tweaking codes is not enough – we must turn back and analyze where we have not yet thought deeply enough, developing a new interdisciplinary **algorithm and performance engineering methodology**. Beyond this, exascale opens fascinating **new opportunities in fundamental research** that go far beyond just increasing the mesh resolution. The opportunities created by **asynchronous algorithms** or **large scale mesoscopic modeling** are just two examples.

---

<sup>1</sup> See a modernized version at:

<http://www10.informatik.uni-erlangen.de/Misc/EIHECS6/Hager.pdf>